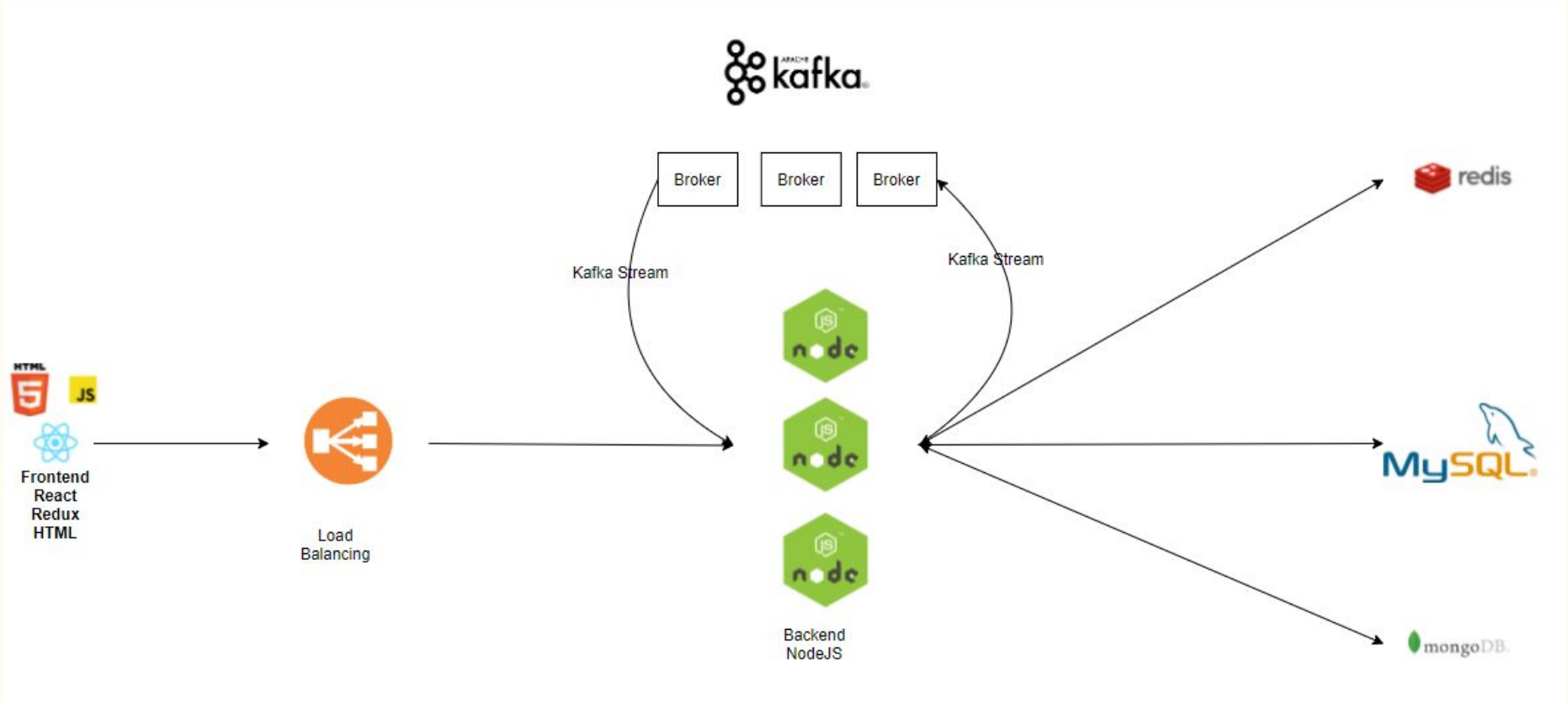
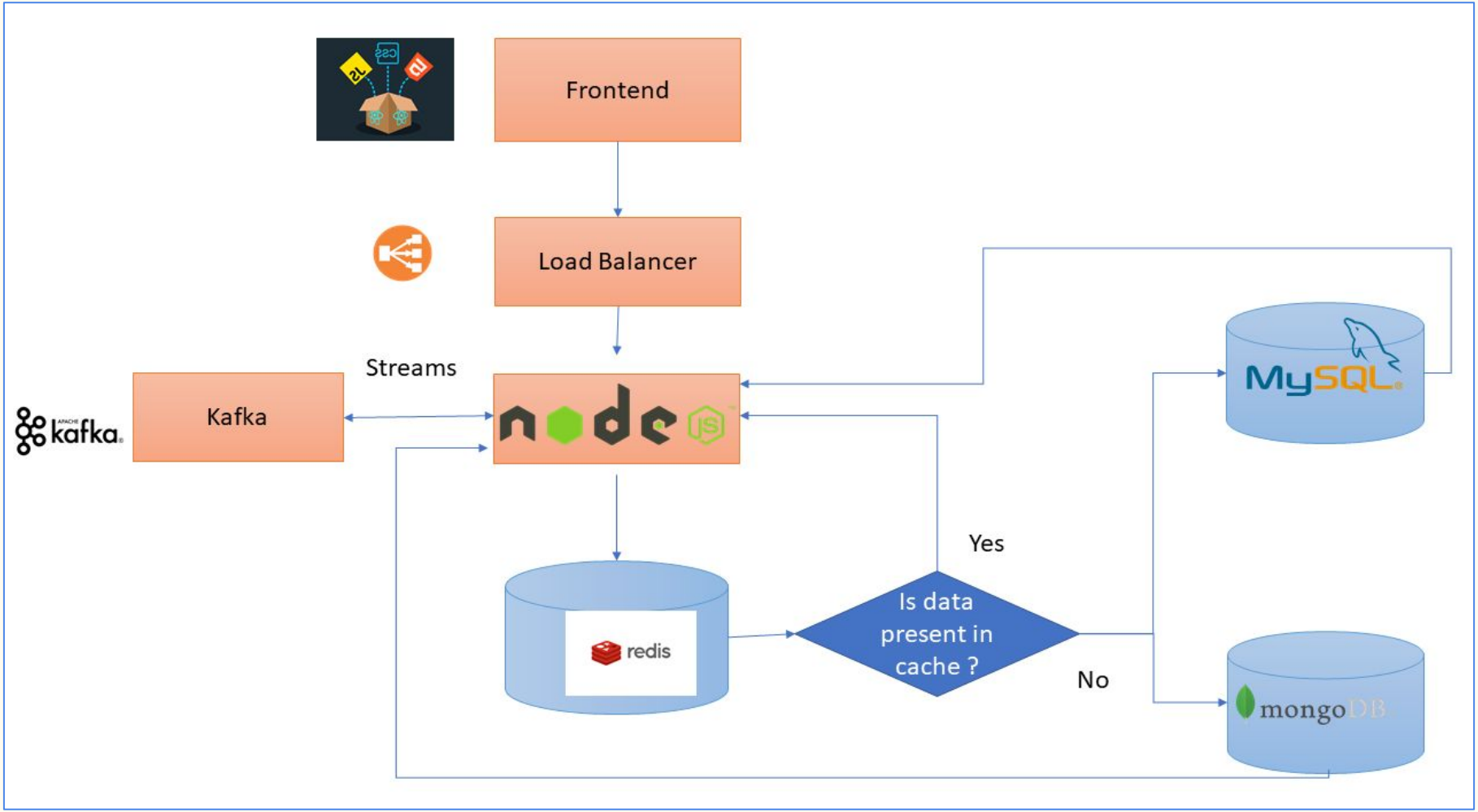
Reddit Clone

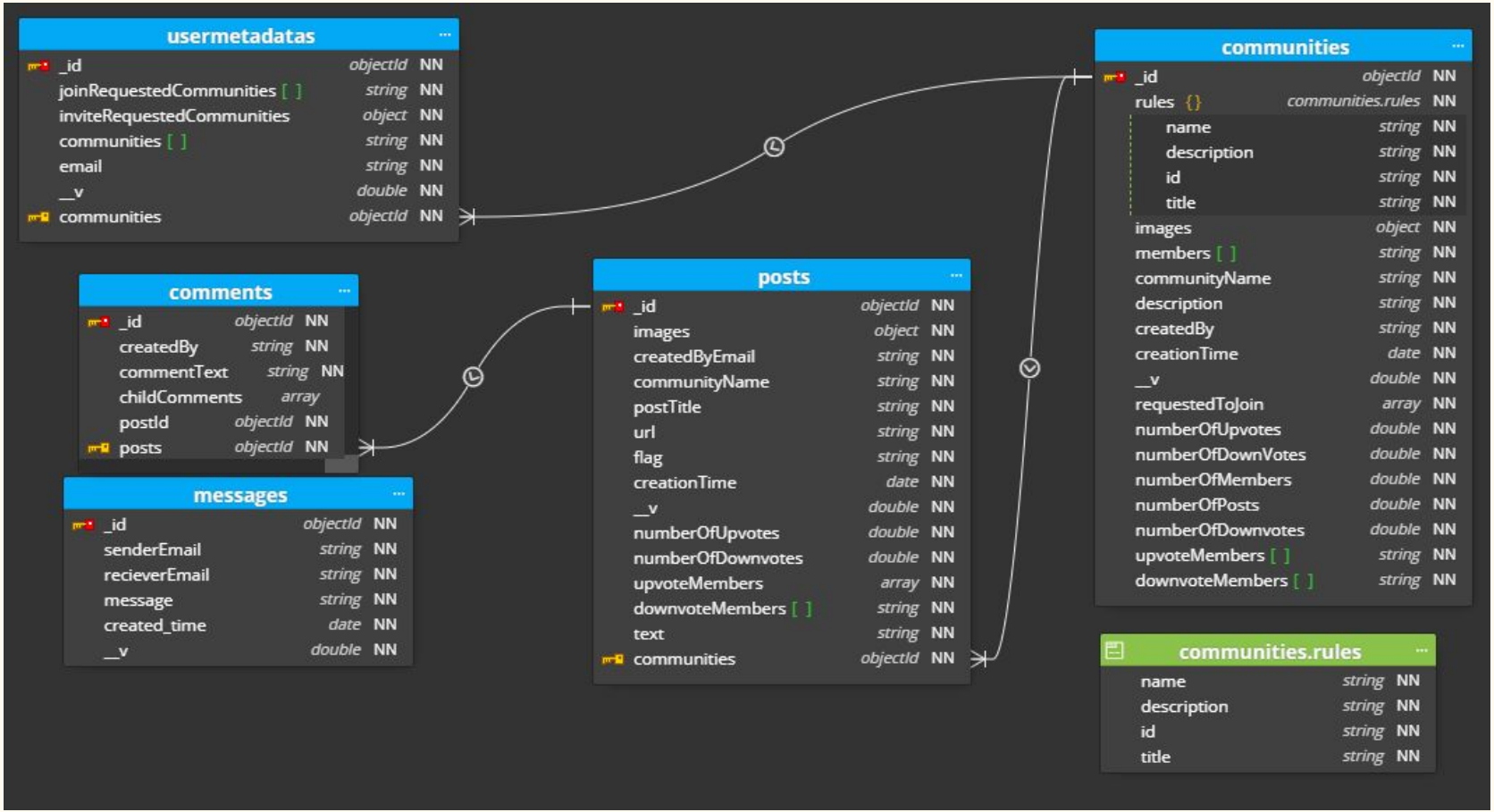
Team 2
-Alihussain Ladiwala -Danesh Vijay Dhamejani -Geetika kapil
-Karan Pinakinbhai Jariwala -Sai Nikhil Yandamuri -Yusuf Juzar Soni

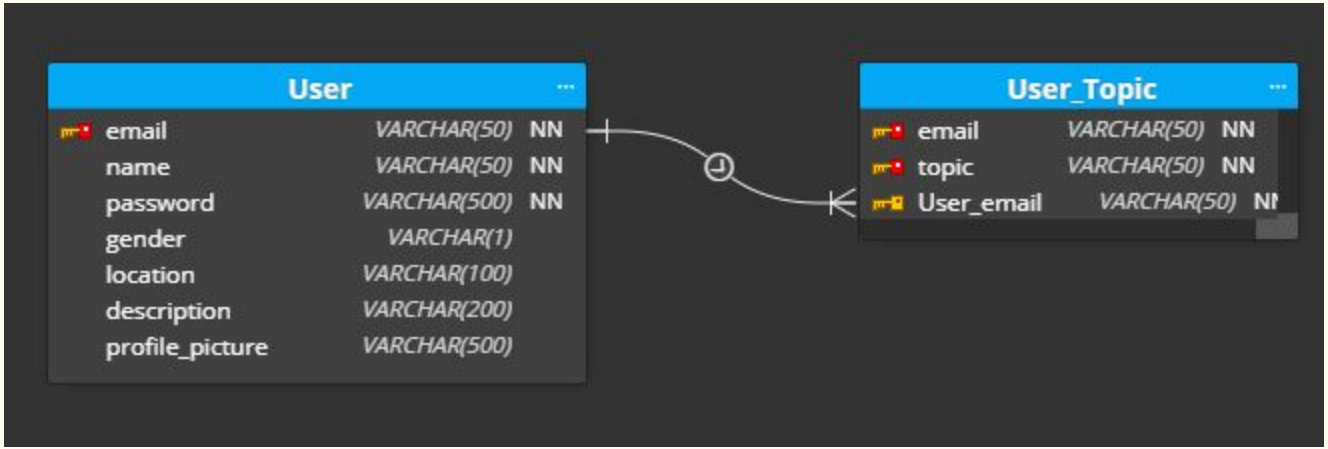
Architecture Diagram




Frontend
Load Balancer
Streams
Kafka
apache kafka
node
MySQL
redis
Is data present in cache ?
Yes
No
mongoDB

ER Diagram


usermetadatas
_id objectId NN
joinRequestedCommunities [ ] string NN
inviteRequestedCommunities object NN
communities [ ] string NN
email string NN
__v double NN
communities objectId NN
comments
_id objectId NN
createdBy string NN
commentText string NN
childComments array
postId objectId NN
posts objectId NN
messages
_id objectId NN
senderEmail string NN
recieverEmail string NN
message string NN
created_time date NN
__v double NN
posts
_id objectId NN
images object NN
createdByEmail string NN
communityName string NN
postTitle string NN
url string NN
flag string NN
creationTime date NN
__v double NN
numberOfUpvotes double NN
numberOfDownvotes double NN
upvoteMembers array NN
downvoteMembers [ ] string NN
text string NN
communities objectId NN
communities
_id objectId NN
rules {} communities.rules NN
name string NN
description string NN
id string NN
title string NN
images object NN
members [ ] string NN
communityName string NN
description string NN
createdBy string NN
creationTime date NN
__v double NN
requestedToJoin array NN
numberOfUpvotes double NN
numberOfDownVotes double NN
numberOfMembers double NN
numberOfPosts double NN
numberOfDownvotes double NN
upvoteMembers [ ] string NN
downvoteMembers [ ] string NN
communities.rules
name string NN
description string NN
id string NN
title string NN


User
email VARCHAR(50) NN
name VARCHAR(50) NN
password VARCHAR(500) NN
gender VARCHAR(1)
location VARCHAR(100)
description VARCHAR(200)
profile_picture VARCHAR(500)
User_Topic
email VARCHAR(50) NN
topic VARCHAR(50) NN
User_email VARCHAR(50) NN

Kafka: Features

CORE CAPABILITIES

HIGH THROUGHPUT

Deliver messages at network limited throughput using a cluster of machines with latencies as low as 2ms.

SCALABLE

Scale production clusters up to a thousand brokers, trillions of messages per day, petabytes of data, hundreds of thousands of partitions. Elastically expand and contract storage and processing.

PERMANENT STORAGE

Store streams of data safely in a distributed, durable, fault-tolerant cluster.

HIGH AVAILABILITY

Stretch clusters efficiently over availability zones or connect separate clusters across geographic regions.

Redis: Features

- **Redis (for REmote DIctionary Server)** is an open source, in-memory, NoSQL key/value store that is used primarily as an application cache or quick-response database.
- Because it stores data in memory, rather than on a disk or solid-state drive (SSD), Redis delivers unparalleled speed, reliability, and performance.
- When an application relies on external data sources, the latency and throughput of those sources can create a performance bottleneck, especially as traffic increases or the application scales.
- One way to improve performance in these cases is to store and manipulate data in-memory, physically closer to the application.
- Redis is built to this task: It stores all data in-memory—delivering the fastest possible performance when reading or writing data—and offers built-in replication capabilities that let you place data physically closer to the user for the lowest latency.

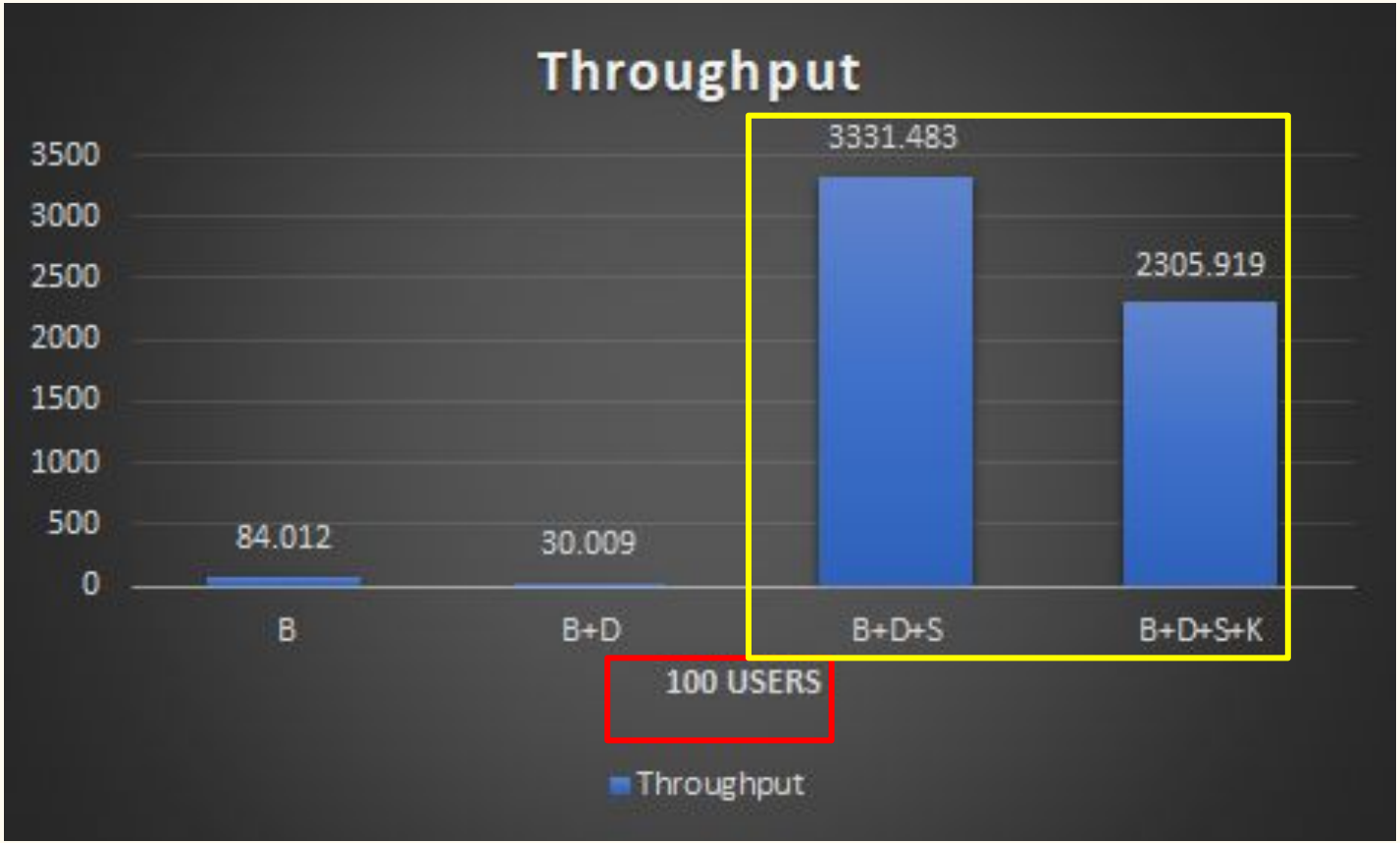
source:https://www.ibm.com/cloud/learn/redis

Performance comparison using different distributed features

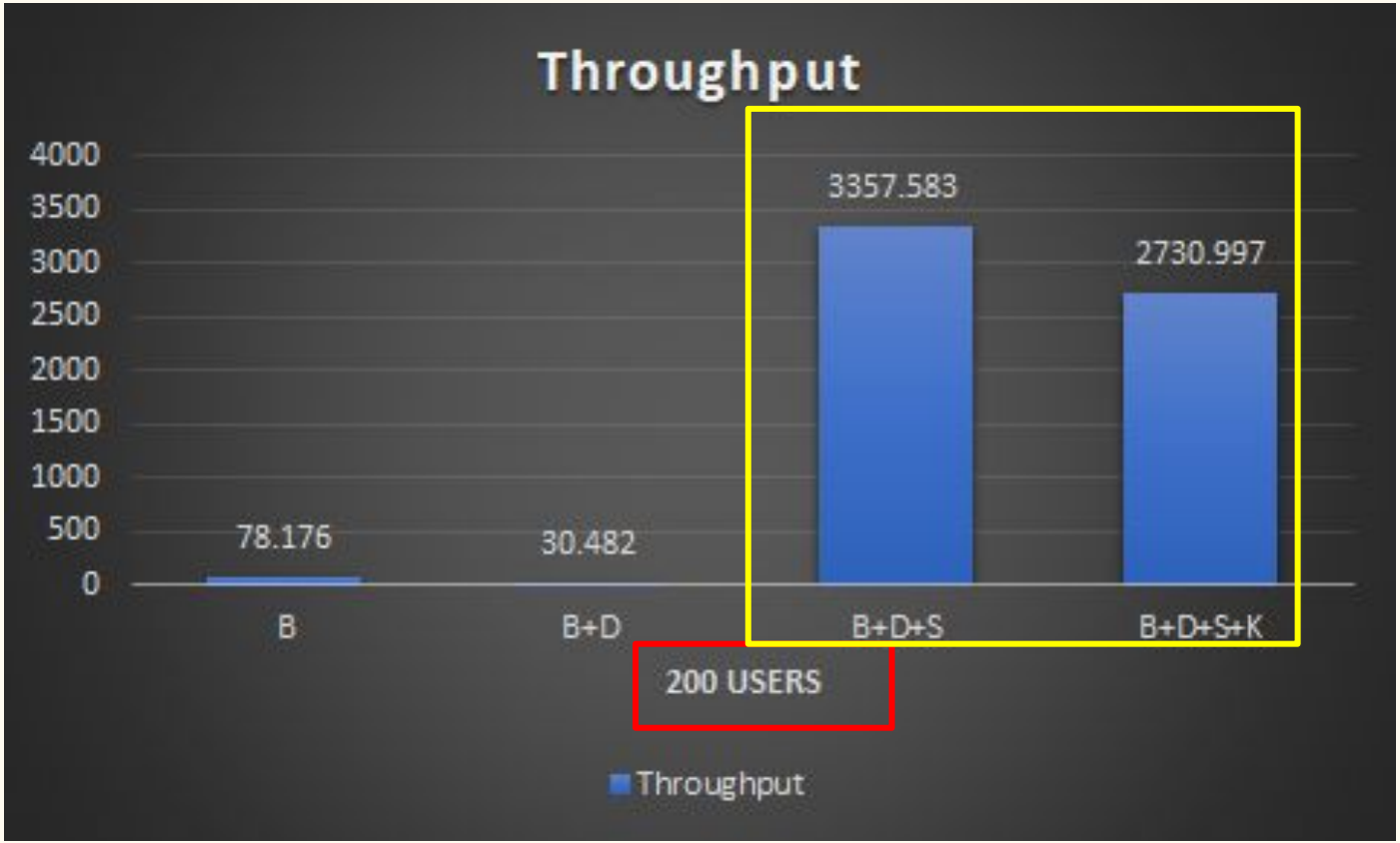
Combinations are:
a. B (Base)
b. B+D (Base+Database Connection Pooling)
c. B+D+S (Base+Database Connection Pooling+SQL Caching)
d. B+D+S+K (Base+Database Connection Pooling+SQLCaching+Kafka

Configurations for Load Balancer
a. 1 Services server
b. 3 Services server
c. 4 Services server

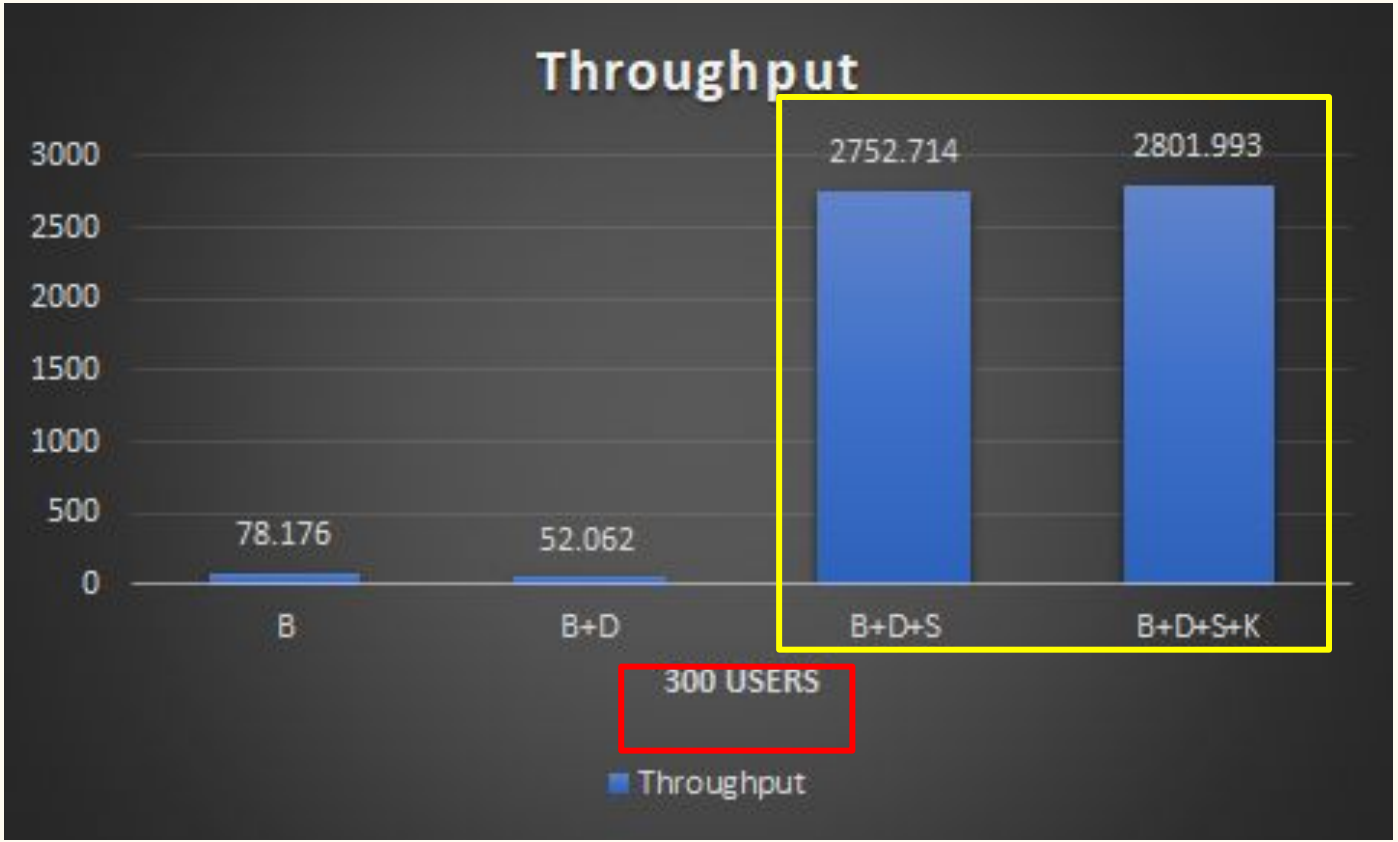
Performance with 100 Users



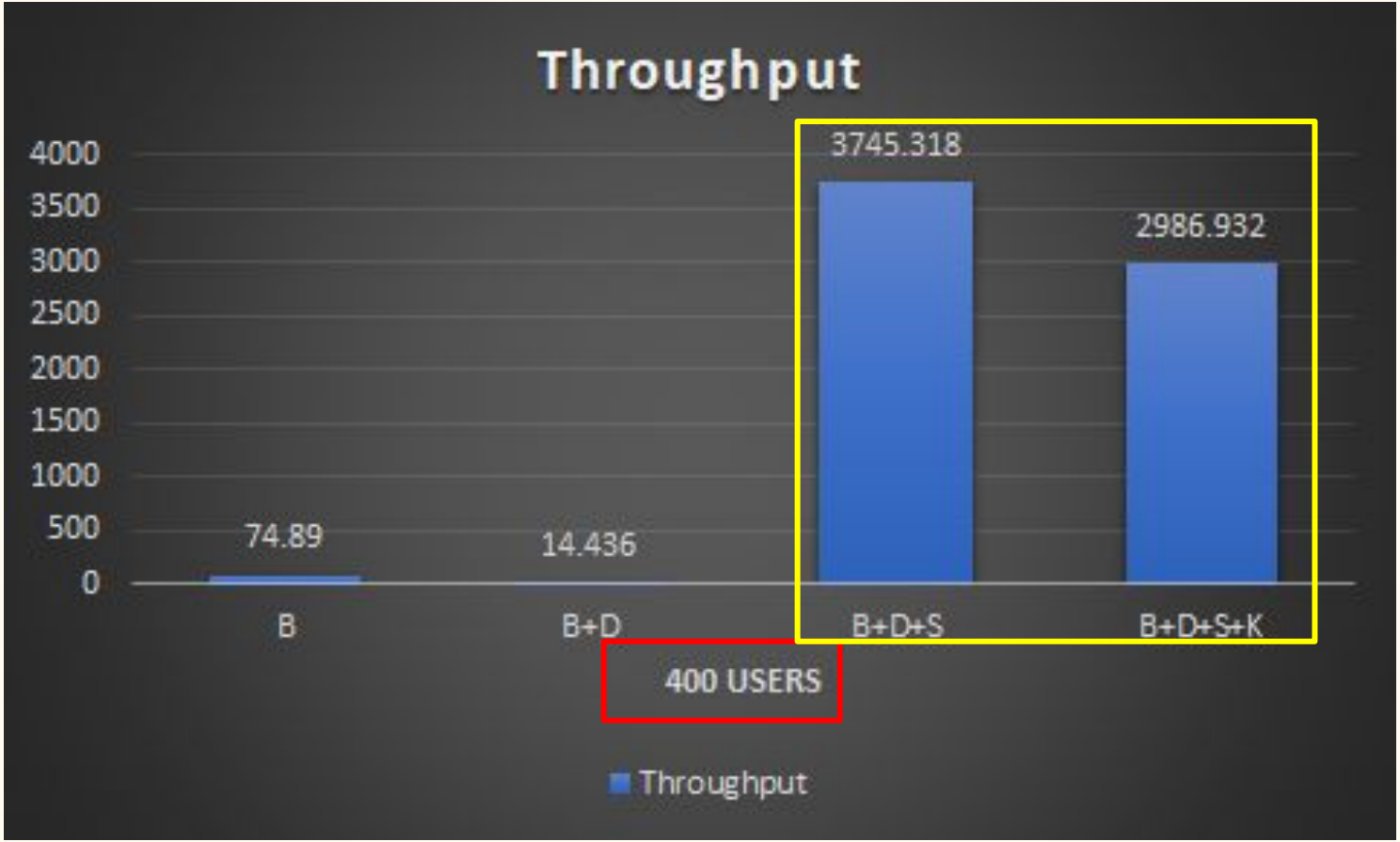
Performance with 200 Users



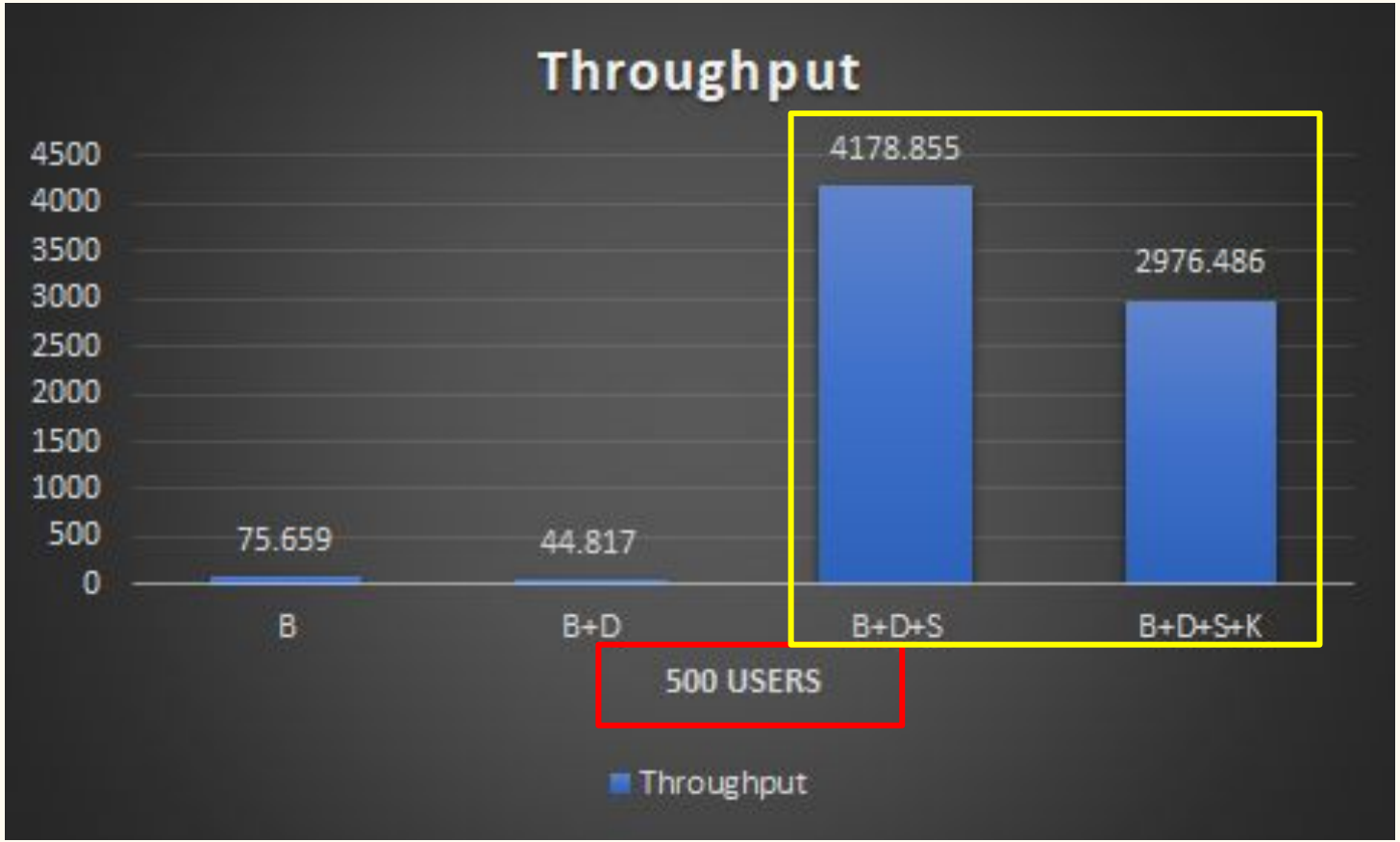
Performance with 300 Users



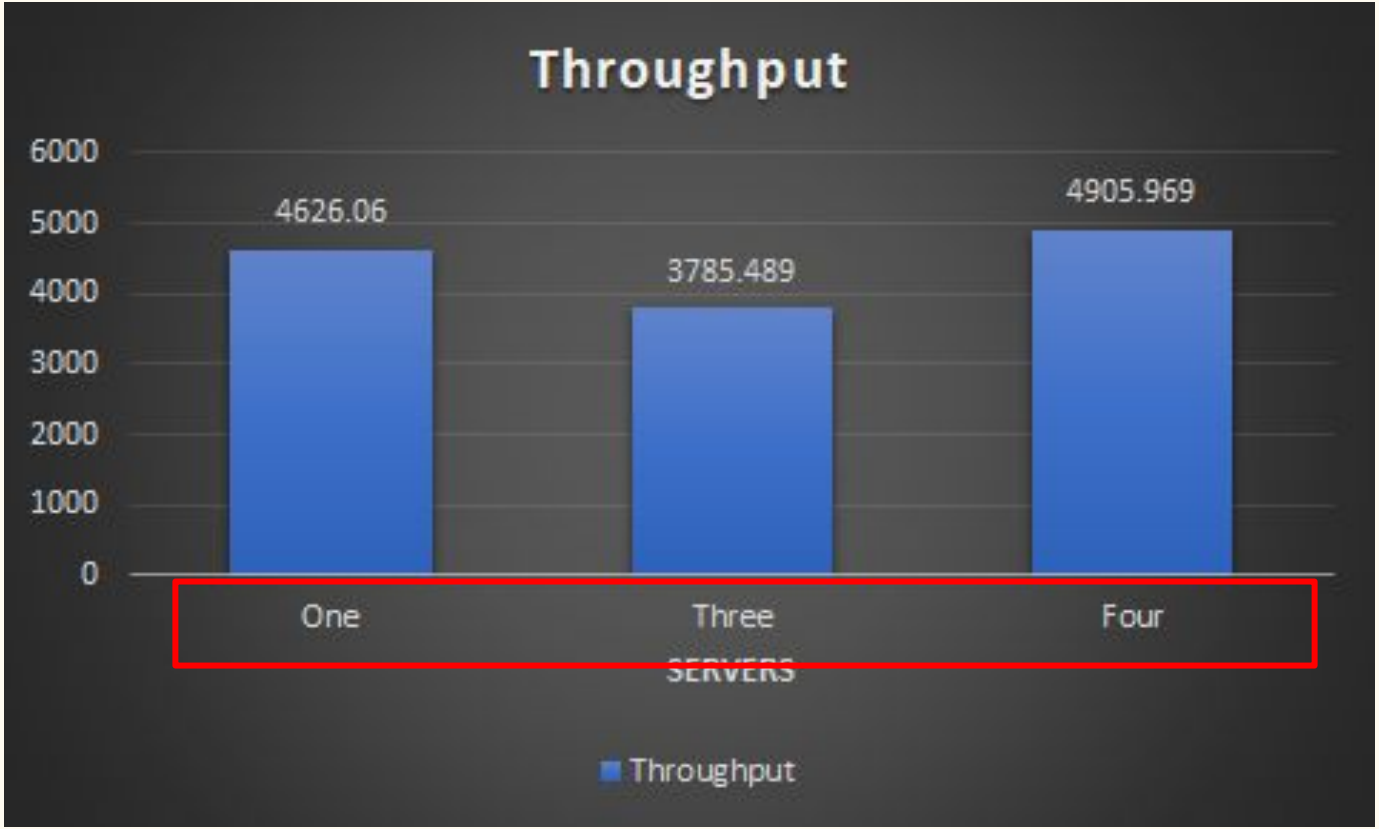
Performance with 400 Users



Performance with 500 Users



Performance Load Balancing



Conclusions

We populated the database with 10,000 messages to test various distributed system strategies.

We have tested the getMessages API with the approaches mentioned in the document.

From the above graphs we can conclude the following

- There is a significant improvement in performance as soon as caching is introduced in the mix.
- Approaches with and without kafka (strategy 3&4)have small difference in performance. Although kafka impacts performance it provides benefits of scalability and high availability.